MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

CENG 111

Fall 2018

Take-Home Exam 4

REGULATIONS

Due date: 20 January 2019, Sunday, 23:59 *(Not subject to postpone)*

Submission: Electronically. You should save your program source code as a text file named `the4.py` and submit it to us via the course's COW page.

Team: There is **no** teaming up. This is an EXAM.

Cheating: Source(s) and Receiver(s) will receive zero and be subject to disciplinary action.

INTRODUCTION

Nowadays, computationally demanding calculations (which normally take weeks or months on a single server) are performed on clusters of computers, where a "controller" computer with low computational resources is provided to the users as a single point of entry. This controller computer receives the "jobs" from the users, maintains a queue of "jobs" waiting to be run, and distributes the jobs to the more powerful computers in the cluster whenever they are available.

A regular interaction with the cluster looks like this:

1. Login to the controller computer.
2. Submit code to be executed on the cluster as a bash script (which arranges all execution, starting from a fresh terminal session; the script handles e.g., nagivating to the right directory, calling the correct executables with correct files as input etc.). The controller then places a "job" request into a queue, and runs it when its turn comes.
3. The user is notified (via e.g., email) when the job finishes execution.

Often, though, the second step fails due to very minor mistakes such as incorrect paths, file/directory names, or bash commands, and the user is notified of these errors only when the job's turn arrives and it is executed, which may take days on a busy cluster.

Your job in this THE, if you accept, is to write functions to identify the issues related to the file system (and related basic commands) before making the user wait unnecessarily.

A File System and Related Commands

An operating system uses a file system to organize and manage files on storage devices. Examples include FAT, NTFS, EXT3, which are widely supported by the Windows and Linux operating systems.

The backbone of a file system can be simplified and represented as an *n-ary* tree, as exemplified in Figure 1, where the nodes correspond to directories and files, and the root node denotes the root of the file system (i.e., "/" on Linux[1]).

[1] In this THE, we will stick to path conventions and commands for the Linux OS.

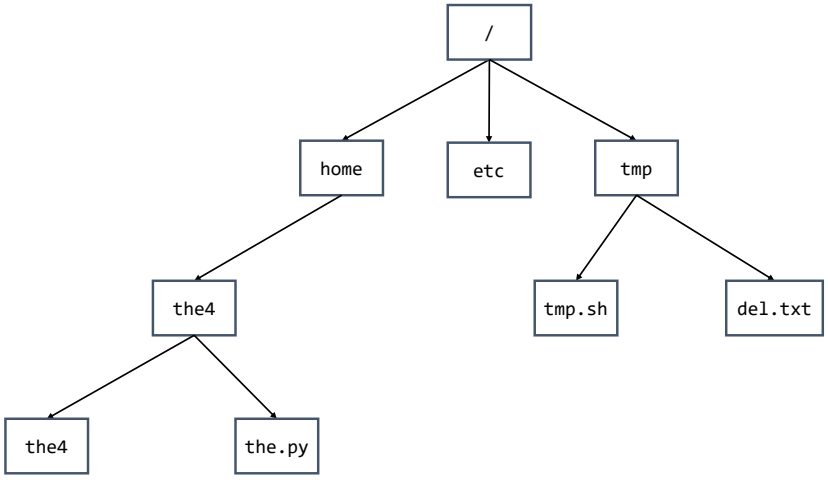

Figure 1: An example file system drawn as a tree.

PROBLEM SPECIFICATIONS

- Write a Python function named `check_commands(FS, C)`, where the first argument (`FS`) describes the file system as a tree and the second argument (`C`) is the list of commands.
- `FS` has the following format for a directory:

  `[name, type, entry-1, entry-2, ..., entry-N]`

  where `name` is a string denoting the name of the directory; `type` is `"d"` or `"D"`; and, `entry-`i represents a subdirectory (having the same format as shown above) or a file with the following format:

  `[name, type]`

  where `name` is a string denoting the name of the file; `type` is `"f"` or `"F"`.

  There can be directories without any sub-directories or files in it. Such directories are represented with `[name, type]`, where `name` and `type` are the same as introduced above.
- File or directory names are case sensitive and composed of letters and numbers only. The length is not limited.
- `C` is a list of strings representing the sequence of commands that the user wishes to execute:

  `[command-1, command-2, ..., command-M]`

- The system allows the following commands:
  - `"cd <path-to-dir>"`: Change the working directory to `dir` if it exists; gives error otherwise. If no argument is provided, it changes the working directory to the root directory (`"/"`).
  - `"mkdir path-to-dir"`: Create a directory named `dir` if it does not exist; gives error otherwise.

  - `"rmdir path-to-dir"`: Remove directory named `dir` if it exists and is empty; gives error otherwise.
  - `"rm path-to-file"`: Remove file named `file` if it exists; gives error otherwise.
  - `"cp path-to-source path-to-target"`: Copies `source` (file or directory) to `"target"` (file or directory). If `"target"` exists and is a directory, `source` (with all its sub-entries) is placed as a sub-entry (if there is already a sub-entry with the same name, `cp` command should return an error). Note that, to simplify your job, this `cp` command is made different from the `cp` command on Linux, which requires providing `"-r"` while copying directories.
  - `"exec file"`: Hypothetically execute `"file"` if it exists and is a file; gives error otherwise.

- Directories and files can be provided to the commands with their paths. A path is a description of the location of the entry in the file system using a set of names. If the entry-name is sufficient to refer to the entry with respect to the current working directory, the path can be omitted and a single entry-name can suffice.

  - A path can be either with respect to the root directory, in which case we call this an absolute path, or with respect to the working directory, in which case we call this a relative path.
  - An absolute path has the following format (note the leading `/`):
    `/dir-1/dir-2/dir-3/.../dir-N/`
  - A relative path follows a similar format:
    `dir-1/dir-2/dir-3/.../dir-N/`
  - A description, `dir-`i, can also be `.` (dot) to denote the current directory, or `..` (two dots) to denote the parent directory.

- The function `check_commands(FS, C)` should return one of the following:

  - `("SUCCESS", FS, WD)` if no error is produced. `FS` is the updated file system following the same format as the input file system, and `WD` is the final working directory.
  - `("ERROR", command, WD)` upon encountering the first error while going through the commands in sequence (the remaining commands are not "executed"); `command` is the full command (as it is provided in the argument) that has caused the error; and `WD` is the final working directory (before the erroneous command).

- While "testing" a command, an error can occur only in the following cases:

  - The command is not one of the following: `cd, mkdir, rmdir, rm, cp, exec`.
  - Insufficient or more-than-necessary parameters are provided to the commands.
  - The argument path for the file or the directory does not exist (unless it pertains to the entry that is to be created by a `mkdir` command) or illogical (e.g., a path includes `..` with respect to the root directory).
  - A `rmdir` command removes a directory that is an ancestor of the working directory; e.g., `rmdir ../../`.
  - A `mkdir` command tries to create a directory whose name matches another directory or file in the same directory.

- A command is trying to delete or overwrite the root directory.
- A `cp` command tries to copy a source that does not exist, or create a file or directory whose name matches another directory or file in the target directory. Let us repeat here the following sentence (from the description of the command above): "If `"target"` exists and is a directory, `source` (with all its sub-entries) is placed as a sub-entry (if there is already a sub-entry with the same name, `cp` command should return an error)".
- A `exec` command is provided a file (or directory) that does not exist; or the argument is a directory.
- In summary: A command should return an error if (i) the syntax is incorrect (incorrect number of parameters), (ii) the paths are invalid as far as the file system is concerned, or (iii) the file/directory described by the path is not suitable for the command (e.g., duplicates, a directory-related command applied on a file etc.).

EXAMPLE RUN

In the following, we provide two simple example runs, which do NOT cover all commands or cases. Note also that the outputs of the functions are manually adjusted to fit and be readable on the page; your functions are not expected to perform this.

```
FS = ["/", "d",
        ["home", "D", ["the4", "d", ["the4", "D"], ["the.py", "F"]]],
        ["etc", "d"],
        ["tmp", "D", ["tmp.sh", "f"], ["del.txt", "F"]]
     ]
C1 = ["cd home/the4/",
        "mkdir ../../home/../home///test///",
        "cd .././../home/test",
        "mkdir /home/test/the4", "cd ..//"]
C2 = ["cd ./home/", "cd ../etc/../tmp/test"]
print check_commands(FS, C1)
('SUCCESS',
   ['/', 'd', ['home', 'D', ['the4', 'd', ['the4', 'D'], ['the.py', 'F']],
                            ['test', 'd', ['the4', 'd']]],
              ['etc', 'd'],  ['tmp', 'D', ['tmp.sh', 'f'], ['del.txt', 'F']]],
  '/home')
print check_commands(FS, C2)
('ERROR', 'cd ../etc/../tmp/test', '/home')
```

NOTES

- The initial working directory is assumed to be root directory (`"/"`).
- There will be at least the root directory in the file system.
- Command, file and directory names are case sensitive.
- The following are valid paths: `/./.home/the4/////`
  `/home/../home/../home/../home/`

- If a command leads to an error, the working directory or the file system should not be affected by the command. However, if a command leads to an error, the effects of the previous commands (that were successful) can stay.
- We will not test your solution with syntactically invalid command/entry names (i.e. names that include non-alphanumerical characters) or entry types (i.e. anything other than `"f", "F", "d", "D"`).
- You are not allowed to import any modules.

HINTS

- A lot of functionality is shared among the different commands. You are advised to decompose your solution into functions, lots of functions.
- Be aware of the aliasing problem. In some cases, it will be helpful for you, and in some cases, it will lead to wrong results. Especially be careful with the `cp` command.

GRADING

- Comply with the specifications and do not print anything on the screen. Since your returned results will be evaluated automatically, non-compliant results will be considered as incorrect by our evaluation system.
- Your program will be tested with multiple data (a distinct run for each data).
- Any program that performs only 30% and below will enter a glass-box test (eye inspection by the grader TA). The TA will judge an overall grade for your solution in the range of [0,30]. The glass-box test grade is not open to negotiation nor explanation.
- A program based on randomness will be graded zero.